Core Principles of Programming: A Learning Blueprint

Master the fundamentals of programming using flowcharts, pseudocode, and the power of AI assistance. Each lesson combines conceptual learning with guided queries to Gemini/ChatGPT, ensuring a language-agnostic approach and reinforcing concepts through real-world examples.

Key Conceptual Points

- **Focus on Fundamentals**: This approach prioritises a deep understanding of core programming concepts for long-term success.
- **Adaptable Mindset**: You'll develop a flexible problem-solving mindset applicable to various programming languages.
- **Procedural Mastery**: We'll establish a strong foundation in procedural programming before progressing to object-oriented and functional paradigms.
- **Visual Learning**: Learning will centre on flow diagrams and pseudo-code for clear logic visualisation.
- **Language Examples**: Practical demonstrations in Python and Java will illustrate concepts.

Key Technical Points

- **Interactive Learning**: Content generation will be supported by ChatGPT or Gemini, with my guidance on targeted questions.
- **Independent Exploration**: Each topic requires self-driven research and expansion for deeper understanding.
- **Progress Checks**: Chapters will integrate exercises, examples, and quizzes to reinforce your knowledge.

How to use this guide:

1. **Open ChatGPT/Gemini**. I would suggest both as they might provide different points of view for the same concept
2. **For each chapter**
    a. Ask the suggested questions to ChatGPT/Gemini;
    b. Read careful the content;
    c. Ask for explanation if something is not clear;
    d. Stop and reflect on what's just read;
    e. Move to the next suggested question;
    f. [Repeat from point a. to e. till you finish the suggested questions]
    g. Do a self assessment using the "Knowledge check" section;
    h. If che chapter has an exercise section, do the suggested exercises and check the results asking ChatGPT/Gemini;
    i. If you can clearly reply to the self assessment questions and have no doubt on the exercise results, move to the next chapter. If not, start from the point you are less confident about;

3. **Explore on your own**, is there any aspect you are more intrigued about, do more research. Always ask why and how?

- History of computers
  - Gemini/ChatGPT Queries
  - Knowledge Check
- Base of electronics
  - Gemini/ChatGPT Queries
  - Knowledge Check
- Operative Systems (OS) and user interfaces
  - Gemini/ChatGPT Queries
  - Knowledge Check
- Networking concepts
  - Gemini/ChatGPT Queries
  - Knowledge Check
- Numeral systems and Boolean logic
  - Gemini/ChatGPT Queries
  - Exercise
  - Knowledge Check
- Introduction to computational logic
  - Gemini/ChatGPT Queries
  - Exercise
  - Knowledge Check
- Flowcharts and pseudocode
  - Gemini/ChatGPT Queries
  - Exercise
  - Knowledge Check
- Variable, data types and the assignment statement
  - Gemini/ChatGPT Queries
  - Knowledge Check
- Sequential control flow
  - Gemini/ChatGPT Queries
  - Exercise
  - Knowledge Check
- Conditional statement
  - Gemini/ChatGPT Queries
  - Exercise
  - Knowledge Check
- Iteration statement
  - Gemini/ChatGPT Queries
  - Exercise
  - Knowledge Check
- Strings
  - Gemini/ChatGPT Queries

History of computers

While you can excel in programming without it, the backstory of how computers came to be is both enlightening and engaging. It's an excellent opportunity to weave in the foundational concept of 'abstraction'.

Gemini/ChatGPT Queries

- Tell me the history of computers, how did we arrive at modern solutions?
- Which are the main changes between each era?
- Which was the first Programmable Machines and how did it work?
- Let's dive deep on the punch card mechanism. That has been used for a while. Can you explain what was the idea behind and how a machine can use that? What was the meaning of punches?
- Which was the next evolution to the punched cards?
- Tell me the evolutions from the punch cards on both input/storage and Programmable machines side.
- Which are the correlation between these two evolutions and the main considerations?
- Is "abstraction" the key driver in this evolution?
- Where does computer evolution go from now on?

Knowledge Check

Can you explain the abstraction concept in computer science?
Can you provide an example of abstraction in computer science?

Base of electronics

This chapter offers a bird's-eye view of electronics, with a special focus on the structure and logic of computer architecture. Grasping how computers operate and 'reason' is key to mastering their instruction and functionality.

Gemini/ChatGPT Queries

- What is electronics?
- What are electronic signals?
- How are electronic signals being used for computer science?
- Why are digital signals used in computers instead of analog ones?
- How are digital signals being processed in a computer? Which basic components are being used?
- Let's dive deep on the high level Computer Architecture. CPU, memory and input/output.
- Explain the main difference between the ARM and x86 classes.
- Explain the various types of memory in a computer.
- Describe what are input/output peripherals and how they work.

Knowledge Check

Can you explain the difference between digital and analog signals?
Can you list the main electronic component of a computer?

Operative Systems (OS) and user interfaces

Understand the core components of an operating system and how they manage your computer's resources. This chapter lays the foundation for understanding how software controls hardware.

Gemini/ChatGPT Queries

- What is an operative system?
- Tell me about the history of OSs.
- What's the main concept of an OS? Explain the "onion" layers concept.
- Explain how an OS interacts with the user.
- What's the difference between a program language and an OS?

Knowledge Check

Can you list the main “layers” of an OS?

Networking concepts

Learn the language of networks and how your applications communicate with the world. This chapter provides a developer-focused look at essential networking concepts.

Gemini/ChatGPT Queries

- How do two computers communicate between each other? explain in simple terms.
- Why would I need this communication to happen?
- Let's dive deep into network protocols, explain the main concepts.
- How does the OSI model work conceptually? Why is it structured in layers?
- Explain a simple simple example of a user requesting a page, like www.google.com, and describe every step mapping it with the relative OSI layer.
- How can I leverage the concepts of the OSI model as a developer?

Knowledge Check

Can you explain what happens when you navigate to www.google.com from your computer?

Numeral systems and Boolean logic

This chapter lays the foundation for understanding how computers work at their most basic level. We'll cover numeral systems and Boolean logic – the building blocks of software instructions.

Gemini/ChatGPT Queries

- What are numeral systems?
- Why are numeral systems relevant for a software developer?
- Let's dig into the binary system, and why it is important in computer science.
- What are logic gates and how do they work?
- Why are logic gates important to understand for developers?
- Let's dig into the boolean logic. Provide examples for the main operations.

Exercise

Convert the binary `0b01011010` into decimal format

Convert the decimal `6749` into Hex format

Perform `0b11001010` **`AND`** `0b01101101`

Perform `0b10111001` **`XOR`** `0b01000101`

Combine AND and OR to extract bits 3 and 4 (zero-indexed) from the binary number `0b1100101`?

Knowledge Check

Can you list the main numeral systems being used?
Do you know how to convert a decimal number into binary?
Can you provide a real example of a boolean logic application?

Introduction to computational logic

Explore the core of software development in this chapter on computational logic. Gain the insights needed to craft sophisticated algorithms and solve complex problems, laying a solid foundation for advanced programming techniques.

Gemini/ChatGPT Queries

- Explain computational thinking and its importance in programming.
- Let's dig into the decomposition and provide more examples relevant for resolving programming problems.
- Let's dig into the pattern recognition and provide more examples relevant for resolving programming problems.
- Let's dig into the ‘abstraction’ and provide more examples relevant for resolving programming problems.
- Explain Algorithms in the computational thinking world.
- Provide some examples of Algorithms in natural language for simple mathematical and non mathematical problems.

Exercise

Create an algorithm, in natural language, for sorting the following list of numbers: `[5, 3, 8, 1, 2]`.
Create an algorithm, in natural language, for finding the most frequent Item in the following list: `["apple", "banana", "apple", "orange", "banana", "apple"]`.
Create an algorithm, in natural language, for generating the first 10 numbers of the Fibonacci sequence (0, 1, 1, 2, 3, 5, 8, ...).
Create an algorithm, in natural language, for checking if a word is a palindrome (reads the same backward as forward), e.g., "radar".

Knowledge Check

Explain the concept of decomposition
Explain the concept of pattern recognition
Can you provide examples of Algorithms?

Flowcharts and pseudocode

Natural language for defining algorithms is good, but there are better ways, which bring us closer to programming languages: Flowcharts and Pseudocode. Let’s have a quick look.

Gemini/ChatGPT Queries

- What are Flowcharts and why do they matter in a programming course?
- List the main components of a flowchart.
- Provide some examples of algorithms in natural language beside the flowchart equivalent representation for simple mathematical and non mathematical problems.
- What is pseudocode and why does it matter in a programming course?
- Provide some examples of algorithms in natural language beside the pseudocode equivalent representation for simple mathematical and non mathematical problems.

Exercise

Create a flowchart of all the algorithms created for the exercises in the previous chapter.
Create the pseudocode of all the algorithms created for the exercises in the previous chapter.

Knowledge Check

Explain the difference between flowcharts and pseudocode.

Variable, data types and the assignment statement

This chapter introduces the fundamental building blocks of programming: variables, data types, and assignment. You'll learn how to store, organise, and manipulate data within your programs.

Gemini/ChatGPT Queries

- What are variables and how are they used in programming?
- Which are the main characteristics of a variable?
- Provide examples of good and bad variable names.
- Explain the concept of "type" for a variable and why it is important.
- List the main variable types with their characteristics and some related examples.
- Let's dig into the strings, explain the concept of strings data type and the main difference from the other primitive types.
- What is the assignment statement and why does it matter for variables?

Knowledge Check

List all primitive data types.
What's the difference between int and float?
How string types are stored in memory?

Sequential control flow

Learn the structure that gives your programs direction. Sequential control flow is the key to designing clear, organised problem solutions.

Gemini/ChatGPT Queries

- Explain what is the "control flow" in programming.
- Let's dig into the "sequential control flow", why is it important?
- Provide some examples of sequential flow in pseudocode and flowchart.

Exercise

Create a program in pseudo code that does 2 at the power of 4 using only sum and the sequential flow. Create a version using one variable and a second version using at least three different variables to calculate the result.

Knowledge Check

What does control flow mean in programming?
When should we use the sequential control flow?

Conditional statement

Learn how to make your programs 'think'. Conditional statements let you write code that executes different sections based on conditions.

Gemini/ChatGPT Queries

- Let's dig into the conditional statement, why is it important?
- Provide some examples of conditional statements in pseudocode and flowchart.
- Can I combine multiple conditional statements together? Provide some examples in pseudocode and flowchart.

Exercise

Create a program in pseudo code that, given a variable with a number (assigned randomly), identifies if it is even or odd.
Create a program in pseudo code that, given a variable with a number (assigned randomly), identifies if it is less than 0, between 1 and 100 or above 100.

Knowledge Check

Explain the difference between sequential and conditional statements.

Iteration statement

This chapter introduces iteration statements (for and while loops). Master the techniques for controlling repetition within your code.

Gemini/ChatGPT Queries

- Let's dig into the iteration statement, why is it important?
- List all types of iterations statements and provide use cases for each one.
- What's the difference between "while" and "do while"?
- Explain the "continue" and "break" concept for iteration statements.
- Provide examples of "for" loops in pseudocode and flowchart.
- Provide examples of "while" loops in pseudocode and flowchart.
- Provide examples of "do while" loops in pseudocode and flowchart.
- Provide examples of loops with the "continue" statement in pseudocode and flowchart.
- Provide examples of loops with the "break" statement in pseudocode and flowchart.

Exercise

Create a program in pseudo code that sums the first 1000 integer positive odds numbers. Create three versions, one with the "for" one with the "while" another one with the "do while" loop.
Create a program in pseudo code that, given two variables with a number (assigned randomly), identifies the lowest common denominator.

Knowledge Check

Explain the difference between "for" and "do while".
List some cases when to use the "break" in a "for" loop.

Strings

This chapter explores strings – the key to handling everything from user input to website content within your programs.

Gemini/ChatGPT Queries

- Why “strings” is such an important type and why does it matter how a programming language handles that?
- Explain at high level how strings are stored in memory and the difference from the primitive types such as “int” or “char”.
- Explain the concepts of Mutability and Immutability for strings, and why it matters knowing the difference.
- Explain what encoding means in the context of “strings” type and more in general in programming.
- Why are there multiple encoding formats?
- Which are the most common encoding formats for strings?

Exercise

Create a program in pseudo code that, given a string variable (with a random word/sentence), identifies which vowel letter is most repeated.

Knowledge Check

Can you explain how strings are stored into memory and why it is possible to directly access one specific letter?

Array and List

This chapter lays the foundation for working with Array and List, a crucial skill for every programmer. They are the backbone of many advanced algorithms and data structures.

Gemini/ChatGPT Queries

- Explain what an array is in a programming language.
- Why and when should I use arrays?
- Provide examples of small programs in pseudocode and flowchart that use arrays of various types.
- What if I don’t have a fixed number of values? Explain the List type.
- Which are the general main differences in terms of creation, processing and memory footprint between arrays and lists?
- Provide examples of small programs in pseudocode and flowchart that use Lists of various types.

- Which alternatives there are to arrays and lists? Provides the differences and use cases examples.

Exercise

Create a program in pseudo code that, given an array of integers (with random values), sort it in ascending order (tip, look for bubble sort, selection sort or alternatives).

Knowledge Check

Explain in simple terms what is an array and what's the difference from a set.
Which data occupies more memory, an array of 10 integers or a list of 10 integers?
Is it faster accessing the latest element of an array or the latest element of a linked list?

Functions

We have been exploring the 'atoms' of software development, the essential building blocks for any programmer. But beyond these fundamentals lie advanced algorithms, data structures, databases, design patterns, and design architectures—exciting topics that you will undoubtedly explore in your future studies. Before we proceed, let's conclude our discussion of the basics with an in-depth look at functions, marking the end of our journey through procedural software architecture.

Gemini/ChatGPT Queries

- Explain what "functions" are in programming languages and how they can be useful.
- Let's dig into their definition, explain the parameters/arguments and the return result.
- Explain what happens when I create and use variables inside a function. Define the variables "scope" concept.
- Explain the various types of parameters, can I use them as variables in the function? What happens when I change them inside and outside the function? Provide examples of the various types and scenarios.
- Provides multiple examples of logic broken down in functions, in pseudocode, for mathematical and non mathematical problems.
- Explain at high level what happens when calling multiple functions, sequentially, in parallel and nested.
- Explain recursion, pros and cons, when I should use and when to avoid. Provide some examples of simple algorithm implemented using recursion vs the same one implemented using iterations statements

Exercise

Create a program in pseudocode that given an array of strings provides all possible permutations.

Knowledge Check

Can you explain what a function is in simple terms?
Can you explain the difference between “by value” and “by reference”?
Explain the difference between recursion and nested calling functions.

Looking forward to your next studies

The journey thus far has introduced us to the core principles of software programming, indispensable for anyone aspiring to master the art. But there's more to becoming a software developer than these foundational skills. This chapter will outline the essential next steps: diving into algorithms and data structures, exploring the realms of modern software architectures, and selecting a programming language to apply your burgeoning skills. From here, your learning should never cease. Whether your interest lies in specialising in a particular software solution, such as mobile development, or in becoming a full-stack engineer, your path will evolve with time and experience. Start with what interests you the most.

Gemini/ChatGPT Queries

- List the advanced algorithm and data structure I need to know as a developer.
- Provide possible real cases when some of these advanced algorithms and data structures are required.
- What is object oriented programming and what’s the difference from procedural software?
- Provide high level examples, in pseudocode, of software solutions built using object oriented programming, for both mathematical and non mathematical problems.
- What is functional programming and what’s the difference from OOP and Procedural?
- Provide high level examples, in pseudocode, of software solutions built using functional programming, for both mathematical and non mathematical problems.
- Why and when should I use OOP or Functional programming?
- List the major programming languages being used and requested now, which types are they and what are their main use cases.
- Provide a high level introduction to databases and now are they used in modern software development?
- Provide the types of modern software solutions available, i.e. Web applications, enterprise solutions, SaaS, mobile, embedded, etc.
- Can you describe cloud computing and IoT?
- What is secure code programming and why is it so important?

Knowledge Check

Can you define the path forward for your studying?